第3章

機能分散型マルチコアLSIを設計してみよう

本誌7月号付属FPGA基板に2個のMicroBlazeを実装する

松本康明

ここでは，米国Xilinx社のソフト・マクロのCPU「MicroBlaze」を活用したマルチコアLSIの設計法を解説する．まず，FPGAに複数のプロセッサを実装する際のシステム設計法を解説する．MicroBlazeでは，二つのプロセッサ・コア間で通信を行うための専用インターフェースを利用できる．このインターフェースを使って，2個のMicroBlazeを使ったシステムを設計する．（編集部）

近年設計される非常に多くのシステム（製品）にマイクロプロセッサが使用されています．また，FPGAもシステム設計時の必須部品として，多くのシステムで利用されるようになっています．

この状況に対応するため，FPGAメーカは自社のFPGA向けにソフト・マクロのCPUを開発しました．現在では，FPGAの性能向上やCPUコア自身の改良などにより，処理性能も向上しています．また，FPGA内部にハード・マクロでCPUを組み込んだ製品もあります．用途によっては，汎用マイコンを置き換えることができるまでになっています．

本稿では，FPGA向けに提供されるソフト・マクロのCPUの応用例として，1個のFPGAに2個のCPUコアを実装するマルチコアLSIの設計を行います．使用するCPUコアは，米国Xilinx社の「MicroBlaze」です．本誌2007年7月号に付属のFPGA（Spartan-3E）ボードを利用します．

1．MicroBlazeの概要

今回は，Xilinx社のMicroBlazeを使用します．そこで，MicroBlazeとその開発ツールについて簡単に説明します．本稿執筆時点の最新版はMicroBlaze 6.0です．

表1にMicroBlazeの主な仕様を示します．一般的な32ビットRISCプロセッサの機能を持ちます．また，FPGAの特徴を生かし，一部機能の有り無しを設定できます．

表1　MicroBlaze 6.0の主な仕様

項　目	仕　様
コア・アーキテクチャ	32ビットRISC
命令長	32ビット固定長
命令セット体系	独自
汎用レジスタの構成	32ビット×32本
バス構成	ハーバード・アーキテクチャ（命令，データ分離）
アドレス・バス	32ビット（4Gバイト）
データ・バス	32ビット
データ配置	ビッグ・エンディアン
割り込み機能	コアに割り込み入力1ポート 専用コントローラで割り込み数を拡張可能
メモリ管理機能	MMU未実装
浮動小数点演算	単精度FPUを実装可能（オプション）
キャッシュ機能	キャッシュ・メモリの設定可能（オプション）
その他	プロセッサ・コア間通信用のFSLインターフェースを最大8ポート設定可能（オプション）
	シフト命令を高速に処理するバレル・シフタ回路を設定可能（オプション）

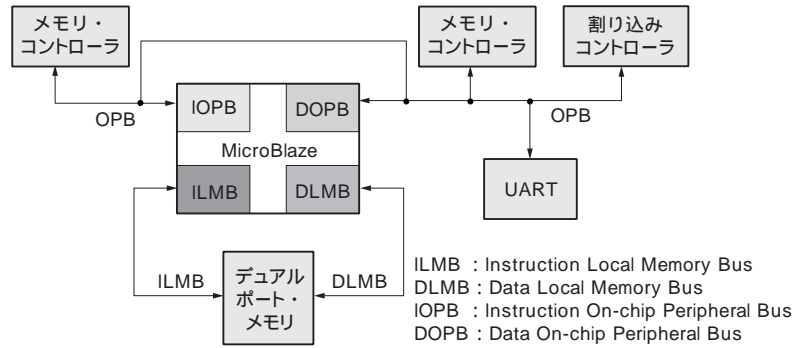
● 2種類のオンチップ・バスを持つ

オンチップ・バスの構成を**図1**に示します．バスとしてLMB（Local Memory Bus），OPB（On-chip Peripheral Bus）の2種があります（それぞれ命令用とデータ用がある）．LMBは主にFPGA内部のメモリ・ブロック（Block RAM）

KeyWord　ソフト・マクロ，マルチコア，MicroBlaze，FPGA，CPUコア，Spartan-3E，オンチップ・バス，LMB，OPB，マルチプロセッサ，共有メモリ，FSL，LCD

図1
MicroBlazeのオンチップ・バスの構成
LMBは主にFPGA内部のメモリ・ブロック(Block RAM)に高速に(2クロックで)アクセスするためのバス．OPBは周辺機能のIPコアを接続するためのバス．いずれも命令用(I)とデータ用(D)がある．

に高速に(2クロックで)アクセスするためのバスです．OPBは周辺機能のIPコアを接続するために使用します．

LMBではデュアルポート・メモリの各ポートをILMB用，DLMB用とすることで，OPBではIOPBとDOPBを接続することで，ハーバード・アーキテクチャのプロセッサで問題となるプログラムを，プロセッサが書き換えられないという問題に対処しています[注1]．

MicroBlazeを利用したシステムの例を**図2**に示します．この例では，LMB上のメモリにプログラムを配置し，DOPB上にUARTやGPIO(汎用パラレルI/O)を接続し，FPGAの外部インターフェースとしています．

● FPGA内蔵メモリ・ブロックを使って動作

MicroBlazeシステムでは，ソフトウェア・コードはLMB上のメモリ・ブロックに配置することを基本としています．しかし，FPGAが内蔵するメモリ・ブロックだけでは，大きな容量をとれません．そのため，LMB上のメモリで容量が不足する場合は，命令用OPB(IOPB)上にメモリ・コントローラを実装し，外部メモリにプログラムを配置する方法をとります．

ソフトウェア・コード用のメモリを実装する場合，LMB用メモリ・コントローラとメモリ・ブロックをペアで使用します[注2]．

Spartan-3Eをターゲットにする場合に使用可能なメモリ容量を**図3**に示します．一つのLMBメモリ・コントローラで実現できるメモリ容量は，8Kバイト，16Kバイト，32Kバイト，64Kバイトの4種類です[注3]．Spartan-3Eの

注1：ハーバード・アーキテクチャのプロセッサでは，命令用バスは読み出し専用なので，そこに置かれるソフトウェア・コードも読み出ししかできないことになる．そのため，ソフトウェア・デバッグなどでコードの書き換えができない．
注2：開発ツールではOPB接続のメモリ・コントローラも用意している．このコントローラで使用できるメモリ容量設定もLMBの場合と同様である．この部分にソフトウェア・コードを配置することも可能だが，読み出し速度(クロック数)がLMBを使用する場合に比べ2～3倍以上遅くなる．
注3：一つのLMBメモリ・コントローラで設定できるメモリ容量は，Virtex-4では最大128Kバイト，Virtex-5では最大256Kバイト．

図2
MicroBlazeを使ったシステムの例
LMB上のメモリにプログラムを配置し，DOPB上にUARTやGPIO(汎用パラレルI/O)を接続している．

Block RAM 使用個数	Spartan-3E使用時のメモリ容量
4	8Kバイト
8	16Kバイト
16	32Kバイト
32	64Kバイト

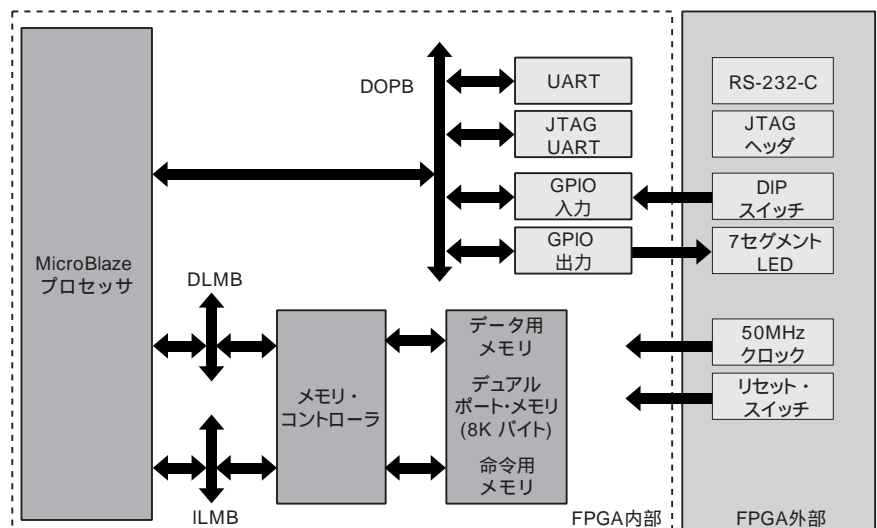
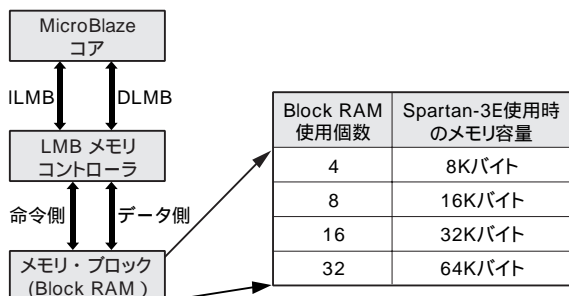



図3　Spartan-3E で使用可能なメモリ容量
一つのLMB メモリ・コントローラで実現できるメモリ容量は，8K バイト，16K バイト，32K バイト，64K バイトの4 種類である．

Block RAMは，1個当たり2Kバイト相当（パリティ付き）の容量があります．従って，例えば64Kバイトの容量を確保する場合は，32個のBlock RAMを使用することになります．

メモリ容量が4段階の構成となっている理由は，**図4**のようにBlock RAMのデータ幅を可変させて（1ビット，2ビット，4ビット，9ビット，18ビットに設定できる），アドレス・バス上に余計なセレクタ回路を入れないように構成しているためです．アドレス入力の前段にセレクタ回路が入ると，この回路の動作遅延がメモリ・ブロック全体の動作速度のネックとなります．

LMBに接続したメモリ・ブロックにソフトウェア・コードを格納する場合，そのデータをFPGAのコンフィグレーション用データとまとめることができます．この場合，FPGAのコンフィグレーションが終了した段階で，LMB上のメモリ・ブロックにソフトウェア・コードがロードされた状態になります．そのため，コンフィグレーション終了後，すぐに動作を開始することが可能です．

● 2種類の開発ツールを使う

MicroBlazeを使用したシステムを設計するには，2種類の開発ツールが必要です[注4]．

EDK（Embedded Development Kit）は，MicroBlazeコアと周辺機能のIPコア，それらをソフトウェアで使用するためのドライバ，マイクロプロセッサ・システムを構築するための各種ツール類，ドキュメントなどで構成されます．Cコンパイラやデバッガも含まれます．

ISEは，FPGA開発ツールです．EDKで作成した回路とHDLで記述した回路を統合し，実際のFPGA上に回路を構成するデータを作成するために使います．ターゲット・デバイスによっては無償版のWebPACKも使用できます．

2．マルチプロセッサ・システムの構成法

近年，システムの高性能化と低消費電力化の要求から，マルチプロセッサ・システムに注目が集まっています．

マルチプロセッサ・システムでは，多くの場合において，あるプロセッサが実行すべき処理が，別のプロセッサの処理結果に依存します．このため，プロセッサ間でデータを共有する何らかのしくみが必要になります．またプロセッサ間でコマンド送受信や状態通知などを行う場合もあります．

注4：EDKにはISEとの間でバージョン依存性がある．EDK内のGUIプログラム（XPS）が起動する際，ISEのバージョン・チェックを行う．このときISEのバージョンがEDK側が期待したバージョンではなかった場合，GUIプログラムは終了してしまう．ISE 9.1iを使用している場合，EDK 9.1を使用する必要がある．

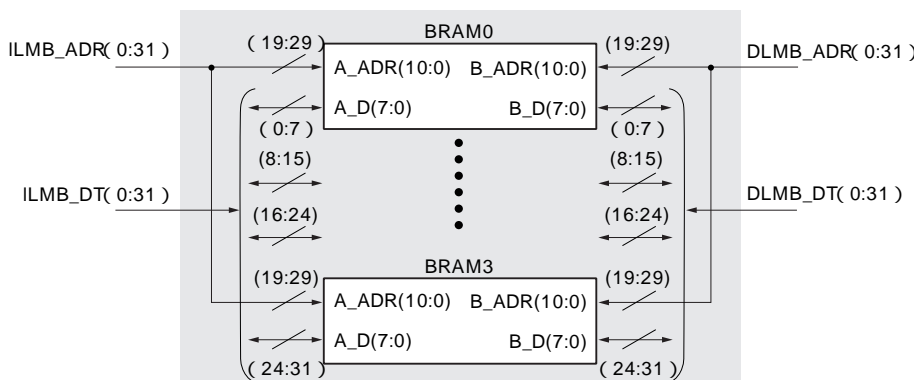



図4
メモリ・ブロックの内部構成
Spartan-3E で8K バイトのLMB 用メモリを構成した場合のBlock RAM の構成を示す．制御信号は省略している．

● 共有メモリによるプロセッサ間通信

MicroBlazeには，性能を追求するマルチプロセッサ構成をサポートする機能はありません．そのため基本的には共有メモリを用います．また，データ共有や通信の際に，共有メモリにデータが書き込まれたことを通知するしくみが必要です．一般的には2種の方法を使用します．

(1) ポーリングによる検出

読み出す側が定期的にフラグを確認します．共有メモリの特定アドレスのデータが，特定のビット・パターンになっていたら書き込みが発生したと判断します．

(2) 割り込みによる通知

書き込み側のプロセッサが共有メモリへのデータ書き込み終了後に，通知すべきプロセッサに対して割り込みを発行する方法です．

● 一つのOPBバスに複数のプロセッサを接続する

一つのOPBバスに複数のプロセッサを接続する構成を**図5**に示します．

この構成では，各プロセッサは自らのLMB上にあるメモリ以外はすべて共有します．各プロセッサはOPBバス・マスタとなり，バス調停はOPB内のバス・アービタがラウンドロビン形式（OPBのオプション設定が必要）で行います．

プロセッサがOPB上の周辺機能（共有メモリも含む）に頻繁にアクセスする場合，バス権の取得待ち状態が発生し，システムの性能（主にプロセッサの処理速度）が低下する可能性があります．また，周辺機能をアクセスする際には，ソフトウェアによる排他制御（セマフォを使用するのが一般的）が必要になります．

プロセッサが3個以上の構成となる場合は，OPBにMicroBlazeを追加します．

● 共有バス上に共有メモリを配置する

共有バス上に共有メモリを配置する構成を**図6**に示します．

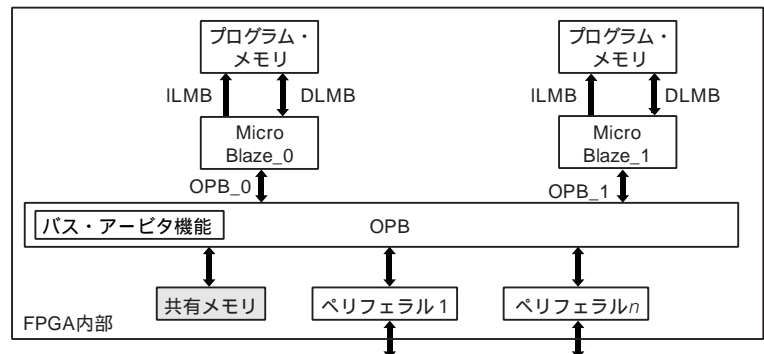
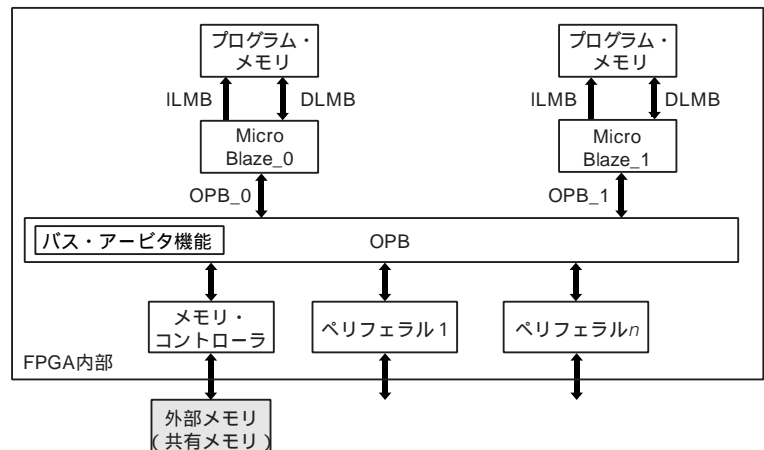
この構成では，各プロセッサは，専用のOPBを持ちます．そしてバス・ブリッジを経由して共有OPB上のメモリと周辺機能をアクセスします．

(a) FPGA内部メモリを共有メモリとして使用

(b) FPGA外部メモリを共有メモリとして使用

図5
一つのOPBバスに複数のプロセッサを接続する構成
各プロセッサは，自らのLMB上にあるメモリ以外はすべて共有する．OPB上の周辺機能（共有メモリも含む）を頻繁にアクセスする場合，バス権の取得待ち状態が発生し，システムの性能（主にプロセッサの処理速度）が低下する可能性がある．プロセッサが3個以上の場合は，OPBにMicroBlazeを追加する．図ではメモリ・コントローラを省略している．

バス・ブリッジを設けることで，一方の専用OPB上で発生するバス・アクセスの影響が他方の専用OPBに影響しなくなります．具体的には，それぞれのOPB上のバス・マスタがMicroBlazeのみのため，バス権取得待ちは発生しません．また，各MicroBlazeの専用OPB上の周辺機能にアクセスする場合，排他制御は必要ありません．ただし，共有メモリへのアクセスは，**図5**の構成と比較してバス・ブリッジなどが増えた分だけ遅くなります．

プロセッサが3個以上の構成とする場合は，追加したいMicroBlazeシステムをバス・ブリッジを介して接続します．

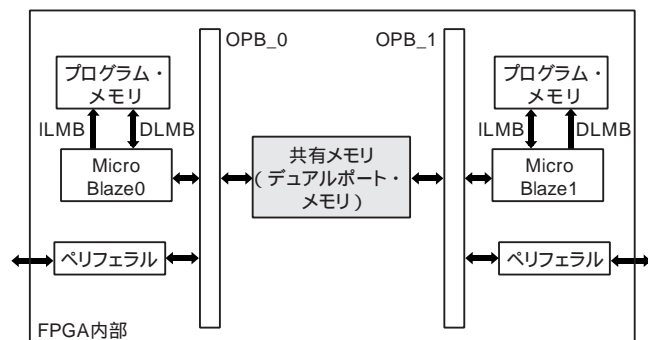



図7　デュアルポート・メモリをOPBに接続する構成

共有メモリは，プロセッサから自身のOPB上の周辺機能として見える．デュアルポート・メモリを使うため，2個のプロセッサ間でしか共有できない．プロセッサが3個以上の構成では，通信を行う2個のプロセッサ間ごとに共有メモリを配置する必要がある．図ではメモリ・コントローラを省略している．

● デュアルポート・メモリをOPBに接続する

デュアルポート・メモリをOPBに接続する構成を**図7**に示します．

この構成では，共有メモリは，プロセッサから自身のOPB上の周辺機能として見えます．バスの共有はありません．従って，共有メモリへのアクセスは，**図5**の構成と同等以上の速度で行えます．

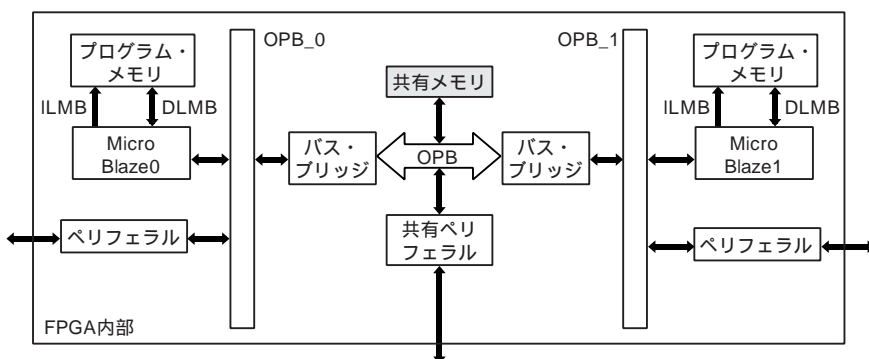
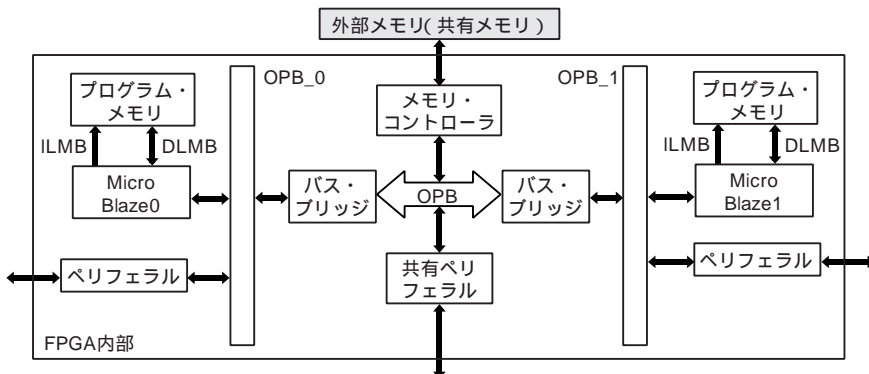
デュアルポート・メモリを使うため，2個のプロセッサ間でしか共有できません．従ってプロセッサが3個以上の構成では，通信を行う2個のプロセッサ間ごとに共有メモリを配置する必要があります．そのため，プロセッサ数が増えるほど，共有メモリの使用数が増加してしまいます．また，3個以上のプロセッサでデータを共有したい場合は，データを書き込むプロセッサがすべての共有メモリに対して同じデータを書き込み，すべてのデータの書き込みが終わったあとで，ほかのプロセッサに通知するといった手順

（**a**）　FPGA内部メモリを共有メモリとして使用

（**b**）　FPGA外部メモリを共有メモリとして使用

図6
共有バス上に共有メモリを配置する構成

各プロセッサは，専用のOPBを持つ．一方の専用OPB上で発生するバス・アクセスの影響が他方の専用OPBに影響しない．共有メモリへのアクセスは，バス・ブリッジなどの分だけ遅くなる．プロセッサが3個以上の場合は，追加したいMicroBlazeシステムをバス・ブリッジを介して接続する．図ではメモリ・コントローラを省略している．

が必要になります．

● 専用インターフェースで接続する

専用インターフェースで接続する構成を**図8**に示します．

この構成では，MicroBlazeコアのオプションとして用意されているFSL（Fast Simplex Link）インターフェースを使います．OPBを使わずにMicroBlaze同士のデータ送受信（いわゆるプロセッサ間通信）を行うしくみです．32ビット長のデータをプロセッサ間で送受信する機能です．この機能を使う場合，FSLインターフェース専用の命令も追加されます．

図9のFSLブロックの中には，データ・バッファ用のFIFOが用意されます． 従って，**図7**の構成に非常に近いことがわかります．プロセッサ・コアから直接出ているインターフェースで，専用の命令でアクセスでき，アクセスが速く，メモリ・ブロックを消費しない（メモリを使用する設定も可能）ところが異なります．

3個以上のプロセッサで構成する場合は，プロセッサ・コアに複数のFSLインターフェースを用意し，ほかのプロセッサと1対1で接続していきます．一つのプロセッサ・コアは，最大8ポートのFSLインターフェースを持つことができます．

3．FSLインターフェースの特徴と動作

FSLインターフェースは，MicroBlazeコアのオプション機能の一つです．EDKで設定が可能です．

● マスタとスレーブで構成

マスタとスレーブが1組で一つのインターフェースを構成します．**図9**に接続例を示します．この例は，データ送信（FSLコア内のFIFOへデータを書き込む）側がマスタで，データ受信（FSLコア内のFIFOからデータを読み出す）側がスレーブになります．

FSLインターフェースを使用する場合は，マスタ・スレーブ間（今回の例では共にMicroBlaze）にFSLコアを挟みます．このコアがデータのバッファリングやバッファ管理を行います．その状態は，Full信号とExists信号で通知します．FSLコア内のFIFOは非同期FIFOとして構成されているため，マスタ側とスレーブ側で使用されるクロックの周波数，位相が異なっていても使用可能です．

データ送信時は，Control信号とData信号を同時に送ります．Control信号は，一緒に送った32ビットのデータが制御用のコマンド・データなのかどうかを受信側が識別するためのものです．ただし，何が制御用のコマンドかを決

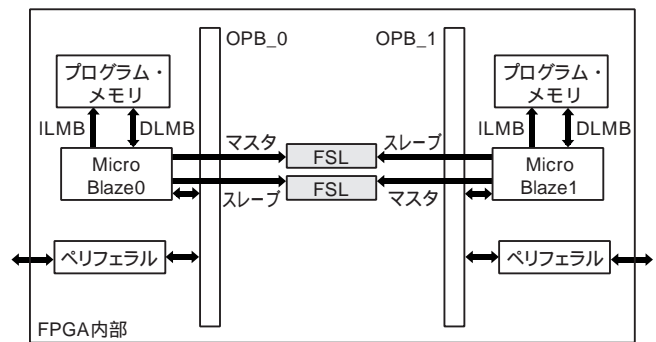



図8　専用インターフェースで接続する構成

MicroBlazeコアのオプションとして用意されているFSL（Fast Simplex Link）インターフェースを使う方法を示す．OPBを使わずにMicroBlaze同士のデータ送受信（いわゆるプロセッサ間通信）を行うしくみである．3個以上のプロセッサで構成する場合は，プロセッサ・コアに複数のFSLインターフェースを用意し，ほかのプロセッサと1対1で接続する．図ではメモリ・コントローラを省略している．

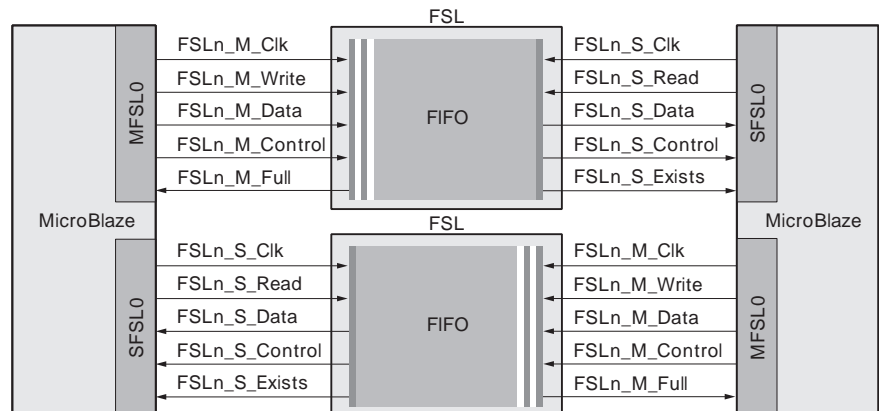



図9
FSLインターフェース

マスタとスレーブが1組で一つのインターフェースを構成する．

定するのはユーザ・ソフトウェアです．ハードウェアでは，制御コマンドか否かを伝えるための信号を用意しているだけです．ちなみにControl＝1（“H”）の場合を制御コマンドの送信としています．

FSLインターフェースのデフォルトの接続では，マスタがデータ送信しても，スレーブ側には通知されません．そのため，データを受信したときに発生する割り込み通知として，FSL_Has_Data信号が用意されています．

FSLインターフェースの読み出しと書き込みのタイミングを**図10**に示します．

● 専用命令でアクセス

FSLインターフェースに対しては，専用のアセンブリ命令でアクセスします．Cプログラム用には専用の関数が用意されています（**表2**）注5．

データ送信は，`put`で始まる命令が基本になります．`putfsl()`関数を実行すると`val`の部分に該当するデータが`id`番号で指定したFSLのポートへ送信されます．このときFSLコアのFIFOがFullの場合，Fullでなくなるまで送信を待ちます．すなわち，プロセッサは`putfsl()`関数が実行完了するまでウェイト状態になります．

データ受信は，`get`で始まるものが基本になります．`getfsl()`関数を実行すると`id`で指定されたFSLポートからデータを受信し，`val`の部分に該当する変数に受信したデータがセットされます．このときFSLコアのFIFOがEmptyの場合，Emptyでなくなるまでデータ受信の状態で待ちます．すなわち，プロセッサは，`getfsl()`関数が実行完了するまでウェイトの状態になります．

FSLインターフェースを操作する関数を使用するためには，mb_interface.hを`include`する必要があります．

注5：アセンブリ命令の詳細については，MicroBlaze Processor Reference Guideを参照．
http://japan.xilinx.com/ise/embedded/edk_docs.htm

表2　FSLインターフェース専用命令

動作	アセンブリ命令	C関数
データ送信	put	putfsl(val, id)
	nput	nputfsl(val, id)
	cput	cputfsl(val, id)
	ncput	ncputfsl(val, id)
データ受信	get	getfsl(val, id)
	nget	ngetfsl(val, id)
	cget	cgetfsl(val, id)
	ncget	ncgetfsl(val, id)

`val`は送信したいデータ（変数も可）
`id`はFSLのポート（0～7）
`n`はウェイトなし
`c`はコントロール制御

表3　文字列変換の動作

制御コード	キー操作	機能
UPPER_CASE	Ctrl＋U	大文字へ変換 abcde ⇒ ABCDE
LOWER_CASE	Ctrl＋L	小文字へ変換 ABCDE ⇒ abcde
NORMAL_CASE	Ctrl＋N	無変換 aBCde ⇒ aBCde
REVERSE_CASE	Ctrl＋R	大文字を入力した場合は小文字へ，小文字を入力した場合は大文字へ変換 abcde ⇒ ABCDE ABCDE ⇒ abcde

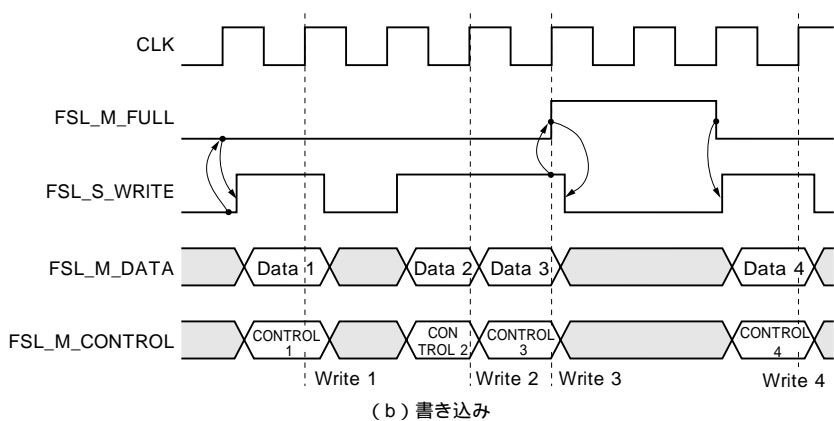
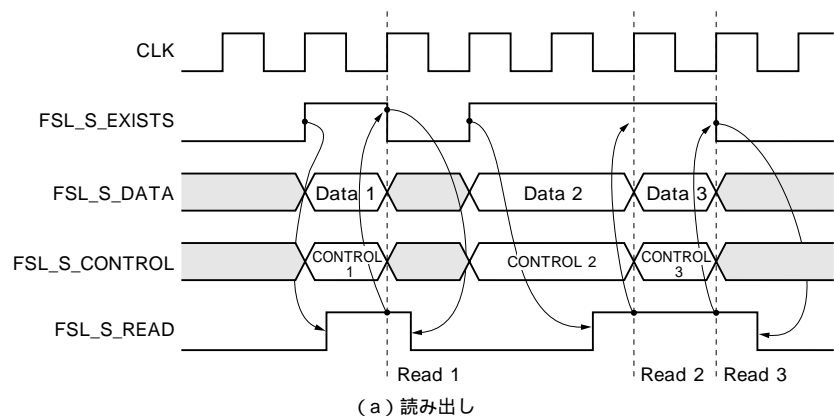



図10　FSLインターフェースの読み出し/書き込みタイミング

4. LCDを持つUARTターミナルを2コアで設計

今回は，FSLインターフェースを使用する簡単なマルチプロセッサ・システムを設計します．サンプルのシステムは，20文字×4行のキャラクタLCDを持つUARTターミナル（通信速度は9600bps）とします．

● 文字列変換を行うターミナル・システム

機能を次にまとめます．

- ターミナルから入力された英数字をあらかじめ設定してある英字の変換ルールに従い，英字の部分のみ変換を行います（**表3**）．
- 変換された文字列をLCDに表示します．また，ターミナルにも文字列を再度表示します．

キーボードから英数文字を入力し，［Enter］キーを押すと，ハイパーターミナルとLCD（20文字×4行）で入力文字が表示されます．英数字入力の前後で制御コード（Ctrl＋1文字）を入力すると，制御コードに従った英字の変換が実行され，変換後の文字がハイパーターミナルとLCDへ出力されます．

LCDで表示できる文字数は80字までです．入力文字数をカウントし，81文字になると入力作業が強制終了します．

● 2個のMicroBlazeで実現する

ハードウェア構成を**図11**に，ソフトウェアの処理フローを**図12**に示します．

今回，UARTの制御はMicroBlaze_0で，LCDの制御はMicroBlaze_1でソフトウェア処理により実行します．

MicroBlaze_0とMicroBlaze_1はFSLインターフェースに接続し，MicroBlaze_0のUARTで受信した文字データをMicroBlaze_1へ送信することと，MicroBlaze_1で文字変換後のデータをFSL経由でMicroBlaze_0で受信することを担います．

● 設計の流れ

実際の設計の流れを説明します．誌面の都合で，ここでは概略のみとします．詳細な手順については，付属CD-ROMに収録したチュートリアル・ドキュメントを参照ください．プロジェクト・ファイルも収録しています．

（1）MicroBlaze_0の作成

プロセッサ・システムを作成するためのGUIツールXPS（Xilinx Platform Studio）を起動します．XPSはEDK内のメインGUIツールです．そして，1個のMicroBlazeとUARTのシステムをEDK内のBase System Builderで作成します（**図13**）．

（2）MicroBlaze_1の作成

もう1個のMicroBlazeとLMB周りの回路を追加します．

BaseSystemBuilderでは，1プロセッサ・システムしか構築できません．そこで，ハードウェアの構成情報が記述されているMHSファイルをテキスト・エディタで編集して，MicroBlaze_1を作成します（**図14**）．

図11 文字列変換を行うターミナル・システムのハードウェア
MicroBlaze_0とMicroBlaze_1は，FSLインターフェースで接続する．FSLインターフェースは，MicroBlaze_0のUARTで受信した文字データをMicroBlaze_1へ送信することと，MicroBlaze_1で文字変換後のデータをFSL経由でMicroBlaze_0で受信することを担う．

制御モードを
NORMAL_CASE
に設定

Input Strings Convert Program Start.Input Settings
[Normal Case Mode]

キーボードから
1文字を入力

総入力文字数は
制限(80文字)
以上？

yes

改行を表示

MicroBlaze_1へ
文字列と
制御モードを送信

MicroBlaze_1
から文字列を
受信

改行して
受信文字列を
表示

no

UARTの受信
バッファから
1文字取得

入力文字は
制御モード？

yes

総入力文字数は
1文字以上？

yes

改行して
全文字列を表示

no

(制御モードを表示)
“ xxx Case Mode ”

no

入力文字は
BS or Del?

yes

一文字削除

no

入力文字は
Enterキー？

no

yes

(**a**) MicroBlaze_0

LCDの初期化設定

関数名
lcdInit
lcdFunctionSet

MicroBlaze_0から文字列
と制御モードを取得

関数名　getfsl

制御モードに従って，
1文字分の変換処理を実行
(英字以外は無変換)

関数名
conv_char_data

変換済み
文字数は
80文字以上？

no

yes

MicroBlaze_0へ
変換後の文字列を送信

関数名　putfsl

LCDへ文字列を書き込み

関数名
displayStrToLCD

(**b**) MicroBlaze_1

図12　文字列変換を行うターミナル・システムのソフトウェア
UARTの制御はMicroBlaze_0で，LCDの制御はMicroBlaze_1で，ソフトウェア処理にて行う．

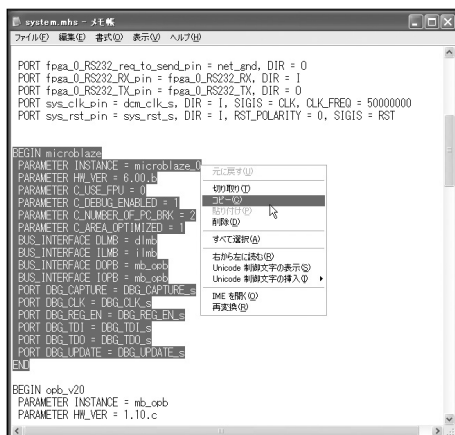



図14　MicroBlaze_1の追加

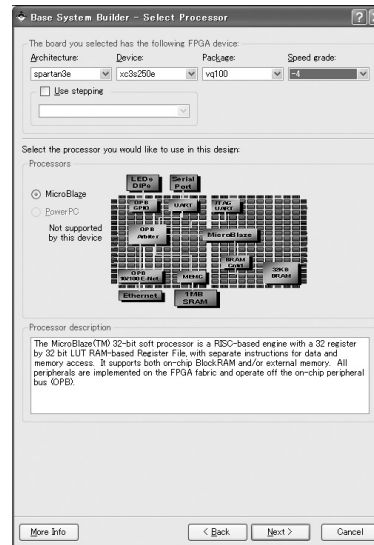
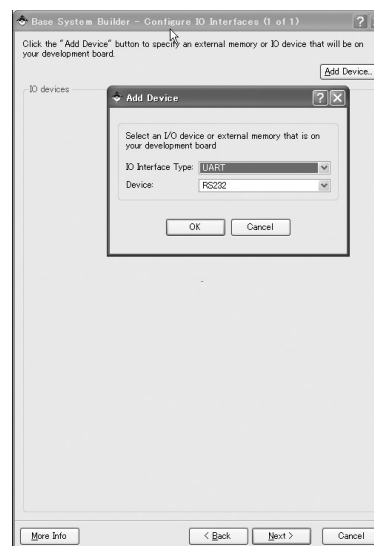
(3) LCD制御回路の設計

LCD制御に使用するタイマを，カスタムIPとして作成します．

ImportPeripheralWizardというGUIツールでユーザ回路記述を含んだ設計ファイルを生成し，それに修正を加えることでバス接続可能なIPコアが作成できます．

さらに，LCD制御用のGPIOをGUIツールで追加します(**図15**)．

(4) ハードウェアの生成

ハードウェアの論理合成・配置配線を行い，FPGAのコンフィグレーション・ファイルを作成します．

① プロセッサの選択

② MicroBlaze の設定

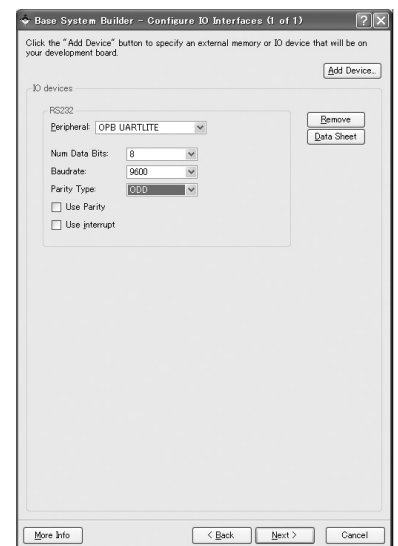
③ UART の追加

④ UART の設定

図13
Base System BuilderによるMicroBlaze_0の作成

(5) ソフトウェアの作成

MicroBlaze_0用のソフトウェアを**リスト1**に，MicroBlaze_1用のソフトウェアを**リスト2**に示します．

二つのソフトウェアをプロジェクトに追加し，コンパイルします．

(6) コンフィグレーション

コンフィグレーション用ファイルとソフトウェアの実行ファイルをマージし，そのファイルをFPGAにダウンロードすれば終了です．

● 7月号付属FPGA基板で動作させる

今回の設計を動作させるための回路図を**図16**に示します．

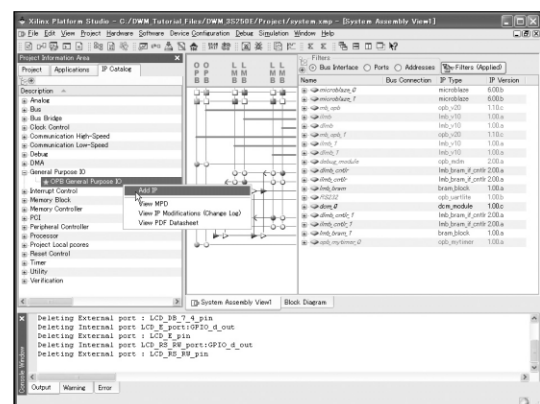



図15　LCD制御回路の追加

リスト1　MicroBlaze_0用ソフトウェア

```
int main()
{  volatile unsigned char rx_data;                    // UARTからの1文字分の受信データを格納
   volatile unsigned char in_char[RX_BUF_SIZE];       // ターミナルからの文字列
   volatile int CaseMode;                             // 英字に対しての変換モード
   int line_position;                                 // 入力中の文字数のカウント
   int line_end_flag;                                 // 1行入力が終了したかどうかの判定フラグ
   int loop;
   unsigned int fsl_rx_data;

   CaseMode = NORMAL_CASE;                            // 変換モードの初期設定

   xil_printf("Input Strings Convert Program Start.\r\n");
      xil_printf("[Normal Case Mode]\r\n\r\n");

   while(1)
   {  line_end_flag = LINE_NOT_END;                   // 1行の入力が終了したかの状態フラグ
      line_position = 0;                              // 文字位置の初期設定
      for(loop=0;loop < RX_BUF_SIZE;loop++)
      { in_char[loop] = END_OF_STRING; }

      while(line_end_flag != LINE_END)                // 1行の入力が終了するまでループ
      {  rx_data = XUartLite_RecvByte(UART_ADR);      // UARTの受信バッファから1文字取得

         if(line_position > LINE_LIMIT)
         { line_end_flag = LINE_END;
           xil_printf("\r\n");
            break;
         }
         else
         {
                           ～中略（入力データに応じた変換モードの変更）～
         }
      }

      putfsl(CaseMode,0);                                                 // 変換モードをMicroBlaze_1に送信
      for(loop = 0;loop < LINE_LIMIT;loop++)  putfsl( ((int)in_char[loop]),0);

      for(loop = 0;loop < LINE_LIMIT;loop++)
      {  getfsl(fsl_rx_data,0);                                           // 変換された文字列をMicroBlaze_1から受信
        in_char[loop] = (unsigned char)fsl_rx_data;
    }

      xil_printf("%s\r\n\r\n",&in_char);
   }
}
```

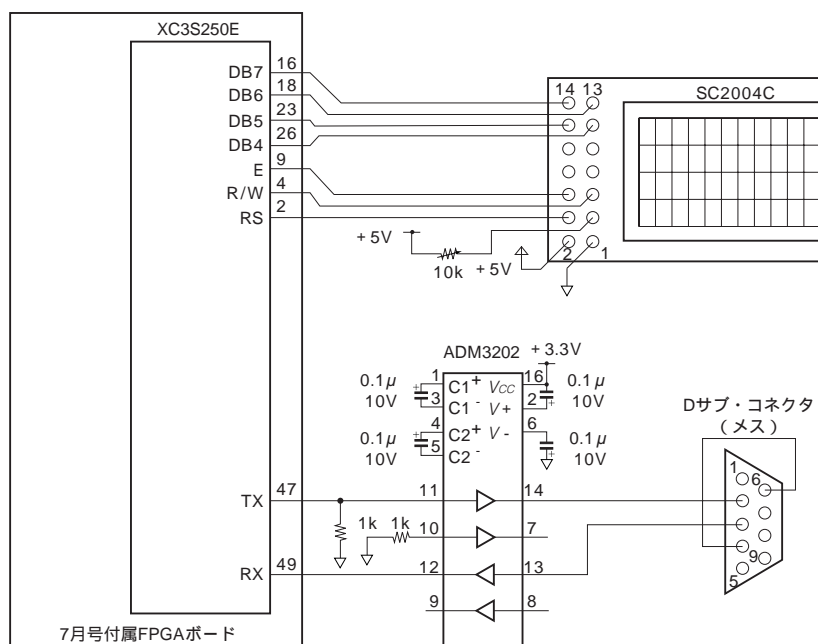



図16
7月号付属FPGA基板で動作させるための回路図

リスト2　MicroBlaze_1用ソフトウェア

```
unsigned char conv_char_data(unsigned char rx_data,int CaseMode)
{
      ～中略（文字の変換処理）～
}


int main (void) {

   volatile unsigned char fsl_rx_data;
   volatile unsigned char in_char[BUF_SIZE];
   volatile unsigned char out_char[BUF_SIZE];
   volatile int CaseMode;
   int in_char_line_position;
   int out_char_line_position;
   int loop;

    lcdInit();
    lcdFunctionSet();

   CaseMode = NORMAL_CASE;

    while(1){

      for(loop=0;loop < BUF_SIZE;++loop)
           {  in_char[loop]  = END_OF_STRING;
              out_char[loop] = END_OF_STRING;
        }

      getfsl(CaseMode,0);                           // MicroBlaze_0から変換モードを受信
      for(loop = 0;loop < LINE_LIMIT;loop++)
      {  getfsl(fsl_rx_data,0);                     // MicroBlaze_0から文字列を受信
        in_char[loop] = (unsigned char)fsl_rx_data;
      }

      in_char_line_position=0;
      out_char_line_position=0;

      while(in_char[in_char_line_position] != END_OF_STRING)
      {  if((in_char[in_char_line_position] >= SPACE) || (in_char[in_char_line_position] < DEL))
         {  out_char[out_char_line_position] = conv_char_data(in_char[in_char_line_position] ,CaseMode);
            ++out_char_line_position;
         }
         ++in_char_line_position;
      }

      for(loop = 0;loop < LINE_LIMIT;loop++)
      {  putfsl( ((int)out_char[loop]),0);          // MicroBlaze_0へ変換後の文字列を送信
     }
      displayStrToLCD(out_char);
    }
}
```

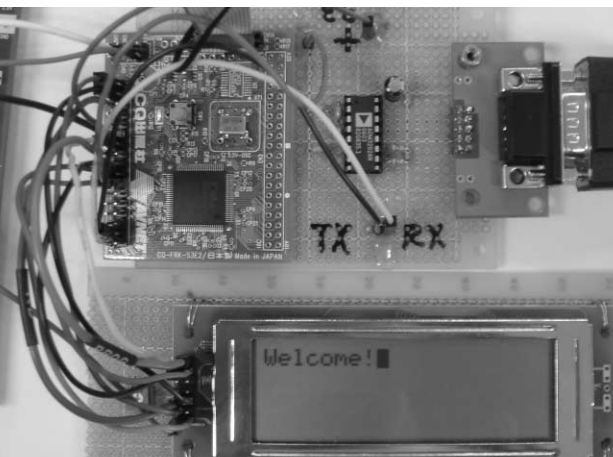



写真1　7月号付属FPGA基板で動作させるために設計した基板

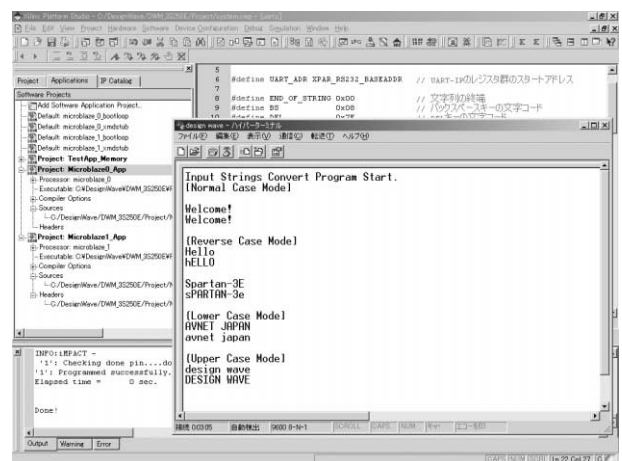



図17　動作の様子

Windowsのハイパーターミナルを使用している．

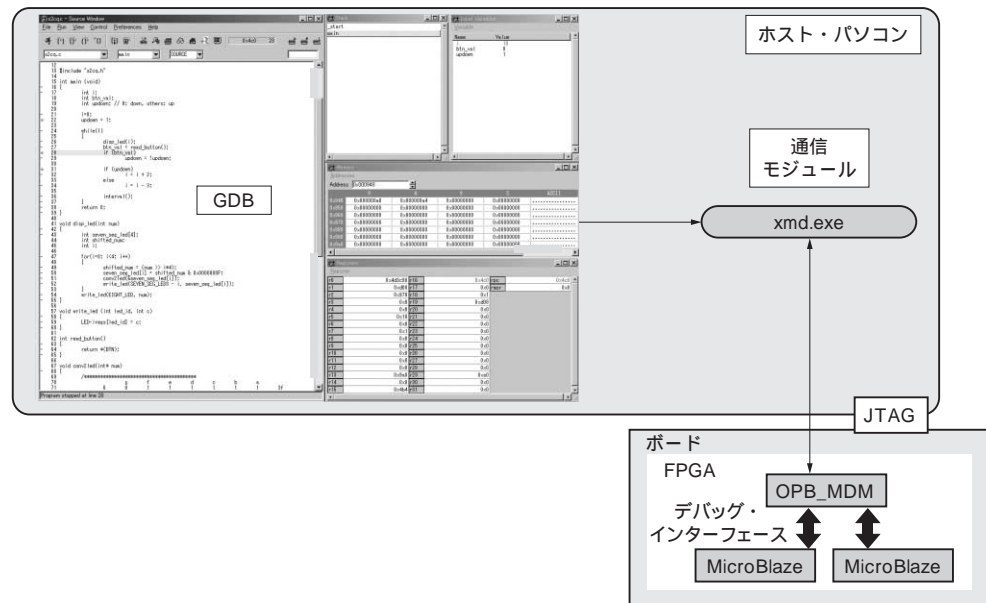



図18
デバッグのためのハードウェア構成

表4　Spartan-3E（XC3S250E）をターゲットにした配線配置結果

項　目	使用数	使用率
Number of BUFGMUXs（クロック・バッファ）	2 out of 24	8%
Number of DCMs（クロック管理ブロック）	1 out of 4	25%
Number of MULT18X18SIOs（エンベデッド乗算器）	6 out of 12	50%
Number of RAMB16s（エンベデッドメモリ）	8 out of 12	66%
Number of Slices（ロジック・リソース）	2273 out of 2448	92%

設計した基板を**写真1**に，動作の様子を**図17**に示します．

Spartan-3E（XC3S250E）をターゲットにした配置配線結果を**表4**に示します．

5．マルチプロセッサ・システムのデバッグ

ここでは主に，EDKに付属のソース・コード・デバッガ（GDB）でMicroBlazeシステムのデバッグを行う方法について説明します．

コラム　付属CD-ROMに収録のデータ

本稿の設計を「実際に試してみよう」，「動かしてみたい」と思われた方のために，付属CD-ROMに設計データを収録しています．

また，ドキュメントとして，Documentsフォルダに，具体的な手順を示したチュートリアルを収めています．

●7月号付属FPGA基板，ISE，EDKをすべてお持ちの方

DWM_3S250Eフォルダ内のProjectフォルダがすべての作業が終了しているプロジェクトです．implementationフォルダ内のdownload.bitをFPGAにダウンロードすると，今回のサンプル設計が動作します．

●7月号付属FPGA基板とISEはあるがEDKがない方

DWM_3S250Eフォルダ内のimplementationフォルダの下にdownload.bitがあります．このファイルをFPGAにダウンロードすると，今回のサンプル設計が動作します．

● 7月号付属FPGA基板を持っていない方

Xilinx社の「Spartan-3AN スタータキット」用にXil_3S700ANを，「Spartan-3E スタータキット」用にXil_3S500Eを用意しました．download.bitファイルをそれぞれのFPGAにダウンロードすると，今回のサンプル設計が動作します．また，EDKをお持ちであれば，実際に設計を体験してみることができます．

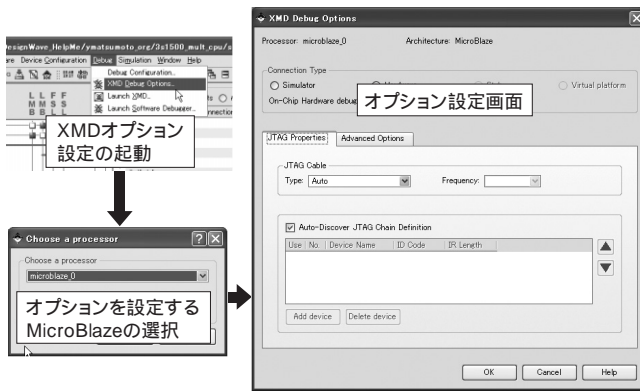
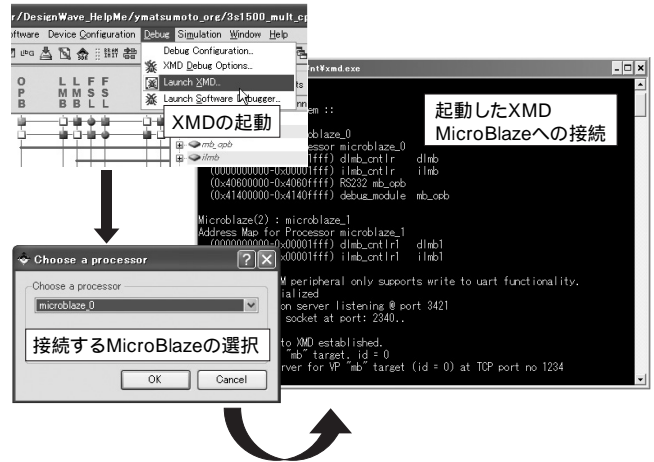
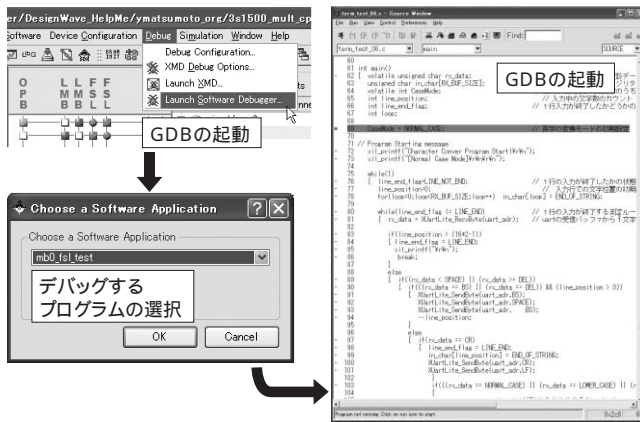
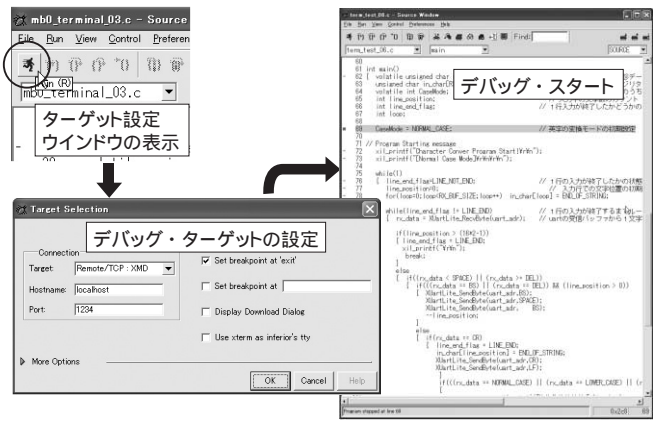
なお，Xilinx社のスタータキットは，アヴネットジャパン（http://www.avnet.co.jp/）およびXilinx社製品取り扱い代理店で入手できます．

（a）XMDオプション設定

（b）XMD起動・MicroBlazeへの接続

（c）GDBの起動

（d）デバッグの開始

図19　デバッガの動作

● デバッグのためのハードウェア構成

まず，ソフトウェア・デバッグを行うためにハードウェアの構成を確認します．FPGA内の接続としてMicroBlazeのデバッグ用インターフェースがOPB_MDMというデバッグ用コアに接続している必要があります．FPGAの外側では，FPGAのJTAGピン，FPGA用書き込みケーブルを介してホスト・パソコン上のデバッガに接続する構成になります（**図18**）．OPB_MDMはMicroBlaze用デバッグ・インターフェースを最大8ポート持つことができます．

● デバッガの動作

実際のデバッグ時は，XMDというプログラムを起動するためのオプション設定後にXMDを起動し，その後GDBを起動する手順になります．操作のイメージは**図19**に示します．

操作の中で重要なことは，デバッグ接続をするMicroBlazeを選択すること，複数のプログラムプロジェクトがある場合，デバッグ対象のプログラム・プロジェクトを選択することの2点です[注6]．

注6：本稿の執筆時点のEDK環境（EDK9.1）では，一度にデバッグ接続を行えるMicroBlazeは，システム上で一つになる．デバッグ接続を行っていないほかのMicroBlazeは，通常の実行を行う．そのため，プロセッサ間のデータ通信部分で予定していない動きをするような場合は，デバッグ接続をしない側のMicroBlazeでデバッグ用のソフトウェアを動作させるなどの工夫が必要になる．

まつもと・やすあき
アヴネット ジャパン（株）